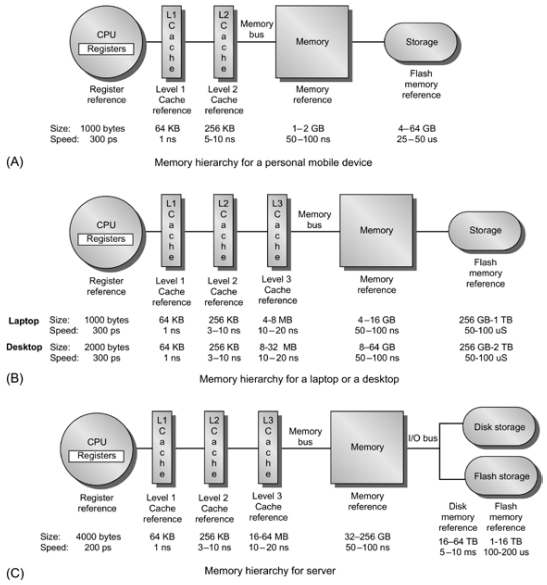
Concepts Introduced in Chapter 2

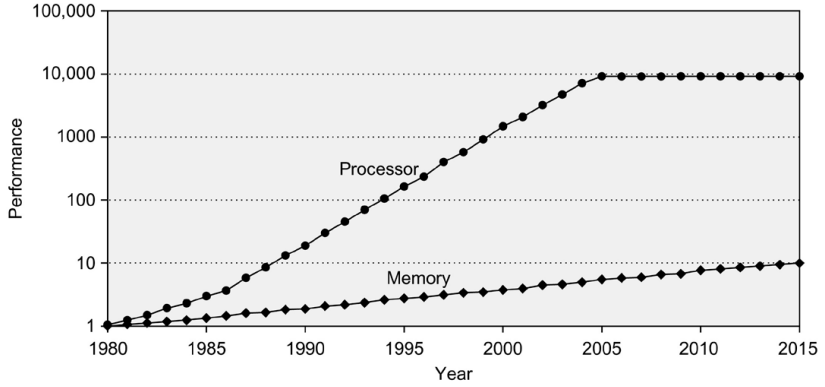
- memory technology and protection
- advanced cache optimizations
- cross cutting issues

Typical Memory Hierarchy Levels



Increasing Performance Gap between Processor and Memory

- Shows relative performance improvements.
- After 35 years main memory is about 10 times faster and processors are about 10,000 times faster.
- In 2005 processor clock rate increases slowed down due to the power wall.



SRAM

- SRAM - Static Random Access Memory
- Used in caches and resides in the same chip as the processor.
- Requires 6 transistors per bit to prevent data from being corrupted when read and requires 4 times the amount of space per bit of information than DRAM.
- SRAM assess time for the L3 cache is about 5 times faster than DRAM.
- SRAM is perhaps 20 times more expensive than DRAM.

DRAM

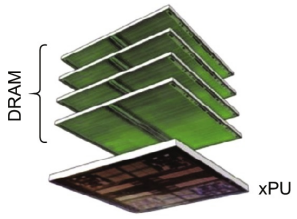
- DRAM - Dynamic Random Access Memory
- Used for main memory.
- Requires a single transistor per bit, which is lost after being read, so each read requires that the data be written back.
- DRAM requires that the data be periodically refreshed, which is less than 5% of the cycles.
- Access time - time between when a read is requested and the desired word arrives.
- Cycle time - minimum time between requests to memory.
- Address pins to DRAM chip are decreased by two by multiplexing the address lines.
- Row Access Strobe (RAS) - first half of the address is sent for the row
- Column Access Strobe (CAS) - second half of the address is sent for the column

DRAM Optimizations

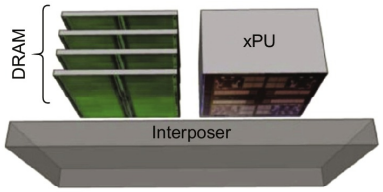
- DRAMs allow repeated accesses to the same row without another RAS, which is called fast page mode.
- Synchronous DRAM (SDRAM) adds a clock signal to avoid overhead of synchronizing with the memory controller and allows a variable number of bytes to be sent over multiple cycles per memory request.
- DRAMs were made wider (in 2017 has 4-bit, 8-bit, or 16-bit memory buses) to simultaneously transfer more bits.
- Double data rate (DDR) transfers data on both the rising and falling edge of the DRAM clock signal.
- SDRAMs introduced 2 to 8 banks that can operate independently and simultaneously service independent requests, which also reduces power.
- SDRAMs have a low power mode, which disables the SDRAM except for the internal refresh. Returning to an active power mode requires about 200 SDRAM cycles.

High Bandwidth Memory (HBM)

- Placing the DRAM and the processor in the same package is called high bandwidth memory (HBM).
- Latency and bandwidth are improved by decreasing the distance required to transmit data and creating more and faster connections between the HBM and the processor.
- One commercially available approach abuts the DRAM dies to the processor with a substrate interposer to contain the connections (2.5D).
- Another approach stacks DRAM dies over the processor (3D).



Vertical stacking (3D)



Interposer stacking (2.5D)

Flash Memory

- Used as the backup storage for PMDs and many desktops.
- Is nonvolatile and uses less power when in standby mode.
- Must be erased before written and is erased in large blocks, often 64KiB to 256KiB.
- There are a limited number of writes for each block, typically at least 100,000.
- Flash is cheaper than SDRAM, but more expensive than disks.
- Flash read accesses are about 150 times slower than SDRAM, but about 300-500 times faster than disk.
- SDRAM write accesses is about 1500 times faster than Flash and Flash writes are about 8-15 times faster than magnetic disk.

Enhancing Dependability

- Larger memories can have errors from the fabrication process (hard errors or permanent faults) or dynamic errors primarily from cosmic rays (soft errors).
- All DRAMs, flash memory, and many SRAMs have spare rows to accommodate errors from manufacturing defects.
- Dynamic errors are detected by parity bits and fixed by error correcting codes (ECCs).

Protection via Virtual Memory

- Virtual memory is also used to provide process protection.
- Page protection bits (read, write, and execute access) are included in each page table (and TLB) entry.
- Only the OS is allowed to update the page table for a process.
- Can switch between a user (user mode) and the OS (kernel mode) by using a system call.
- Additional protection can be provided by virtual machines (VMs).

Basic Cache Optimizations

- Goal is to improve average memory access time.
  - hit_time + miss_rate * miss_penalty
- basic miss rate optimizations
  - larger block size, bigger caches, higher associativity
- basic miss penalty optimizations
  - multi-level caches, giving priority to read misses before write misses.
- basic hit time optimizations
  - virtual caches, virtually indexed and physically tagged caches

Further Improving Caches

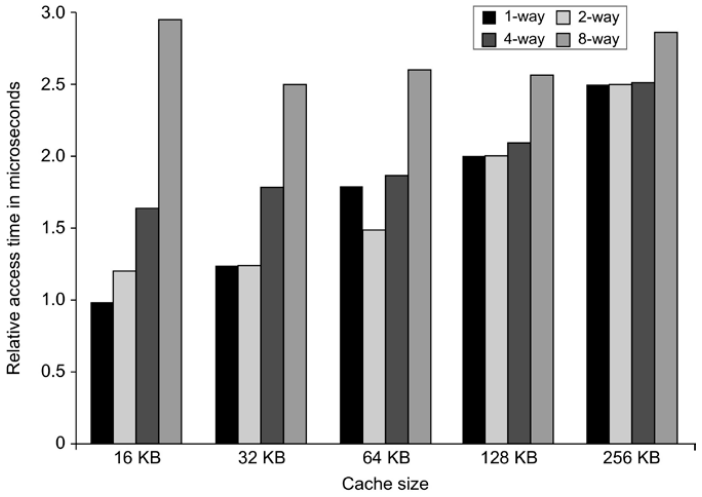
- Also possible to improve cache bandwidth and energy usage.
- reducing hit time
  - small and simple first-level caches
  - way prediction
- increasing cache bandwidth
  - pipelined cache access and multibanked caches
  - nonblocking caches
- reducing the miss penalty
  - critical word first and early restart
  - merging write buffer
  - using HBM as an L4 cache
- reducing the miss rate
  - compiler optimizations
- reducing the miss penalty or miss rate via parallelism
  - hardware prefetching
  - compiler controlled prefetching

Small and Simple First-Level Caches

- faster
  - Less propagation delay due to smaller size.
  - Direct-mapped organization allows the tag check to overlap with the transmission of data to the CPU.
- more energy efficient
  - Smaller size results in both lower dynamic and static power.
  - Lower associativity means fewer tag comparisons and fewer data words accessed on loads.

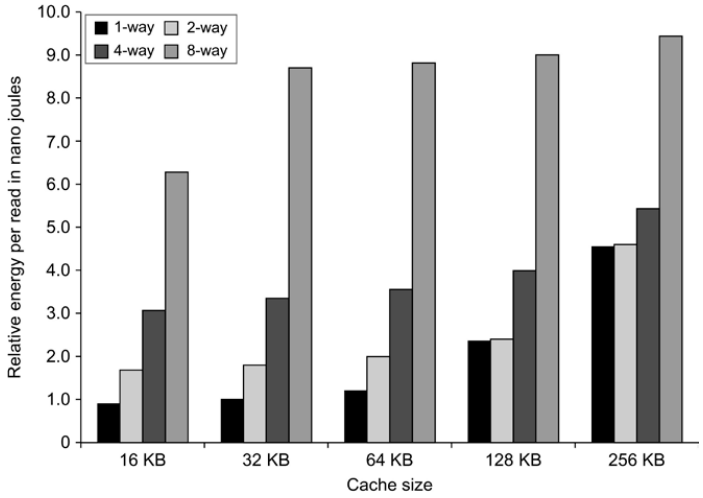
Access Times as Cache Size and Associativity Are Increased

- Less associativity and smaller size is generally faster.



Energy Used as Cache Size and Associativity Are Increased

- Less associativity and smaller size is generally more energy efficient.

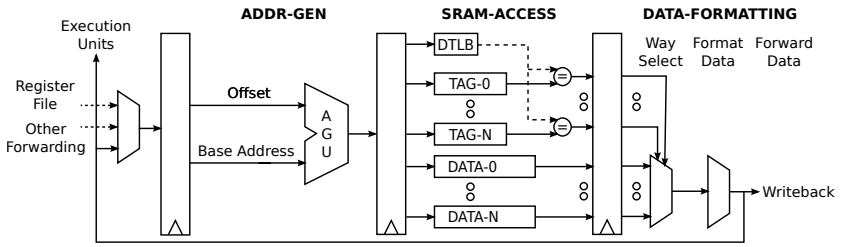


Way Prediction

- In each cache set are predictor bits indicating which block (way) in the set to check on the next access to that set.
- A way hit is about as fast as a direct-mapped organization.
- A way misprediction requires checking the other block(s) in the set on the subsequent cycle.
- Decreases misses as in a set-associative organization.
- Can decrease power due to checking fewer tags and accessing fewer data words in parallel.
- Increases complexity as will have fast and slow hits.
- The miss penalty is also increased by one cycle.

Pipelined Cache Access

- Can simply have the cache access pipeline stages (IF and MEM) take multiple cycles.
- Allows for a faster clock cycle time.
- Typically will result in a greater branch misprediction penalty and increased delay between a load and the first instruction being able to use the data.
- Example of pipelining loads:

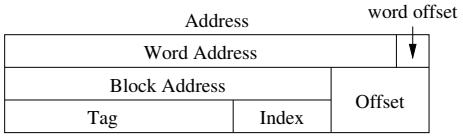


Multibanked Caches

- Divide the cache into *n* independent banks to support simultaneous accesses to different banks and improve cache bandwidth.
- Usually sequential interleaving is applied where the block or word address modulo *n* is used to determine which of the *n* banks to access.
- Provides support for multiple issue processors for both instructions and data.
- The L2 or larger cache levels can also supports simultaneous access of instructions and data in the unified caches when accessing different banks.
- Banking reduces energy usage when only a single bank (as opposed to the entire cache) is accessed for a single instruction or data reference.

Four-Way Interleaved Cache Banks with Block Addressing

- Block interleaving allows both tags and data to be independently accessed.
- Word interleaving would require duplicating the tags for each bank, but can result in fewer bank conflicts when performing sequential accesses.

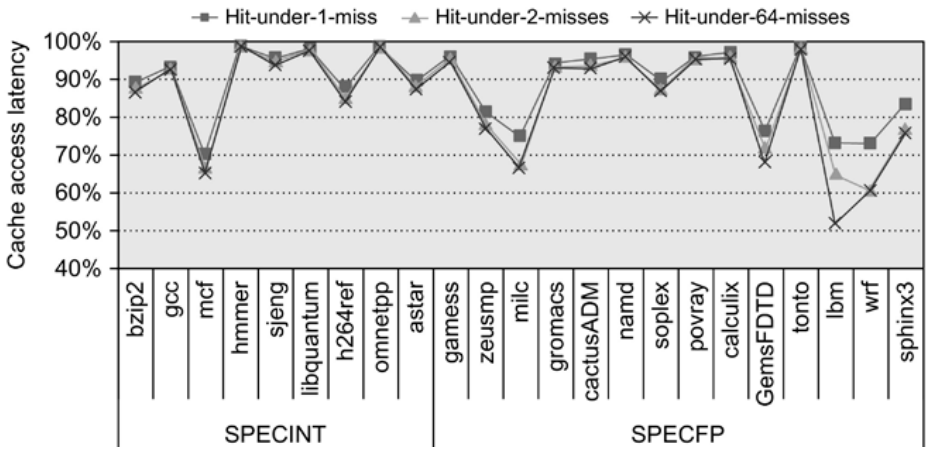


Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

Nonblocking Data Caches

- Allows accesses for cache hits while a cache miss is being serviced on an out-of-order processor.
- Some caches allow a hit under multiple misses, which uses miss status handling registers (MSHRs) to track which instructions caused which misses to allow these instructions to proceed after the block is loaded into the cache.
- Need a status bit for each block to indicate if it is being filled.
- Improves performance by increasing the cache bandwidth, which reduces the effective miss penalty since there are fewer stall cycles.

Average Memory Stall Time with a Nonblocking Data Cache



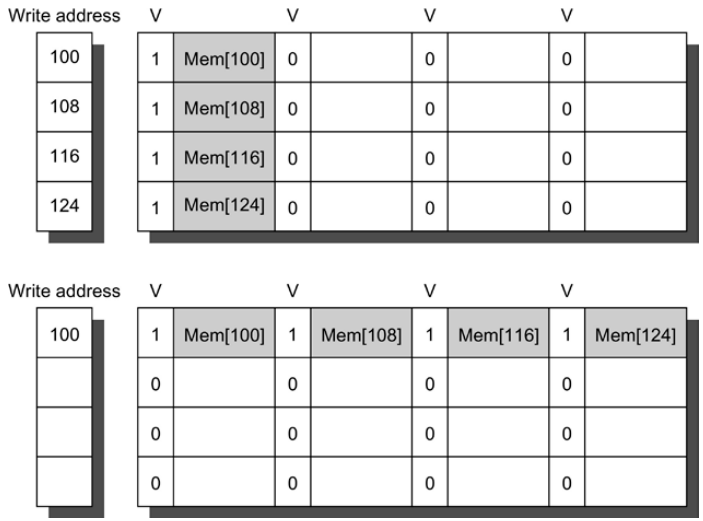
Critical Word First and Early Restart

- Critical word first means to request the missed word first from the next memory hierarchy level to allow the processor to continue while filling in the remaining words in the block, usually in a wrap-around fill manner.
- Early restart means to fetch the words in the normal order, but allow the processor to continue once the requested word arrives.
- Both approaches effectively reduce the miss penalty.
- Status bits must be used to indicate how much of the block has arrived.
- More beneficial for caches with large block sizes.

Merging Write Buffer

- Due to the latency penalty, multiword writes are faster than writing words in separate requests to the next memory hierarchy level.
- Multiple words in the write buffer can be associated with the same block address.
- Valid bits can be used to indicate if a word within the block should be written.
- Reduces the number of accesses to the next memory hierarchy level.
- Reduces the miss penalty due to fewer write buffer stalls for a given buffer size.

Example of Write Buffer without and with Write Merging

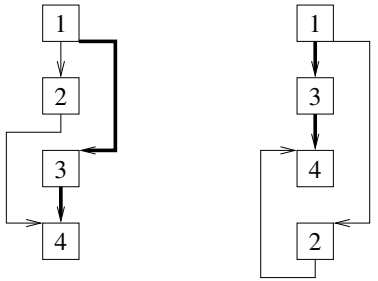


Compiler Optimizations

- Compiler optimizations can be used to reduce the cache miss rate without requiring any hardware modifications.
- Improves spatial locality.
  - code positioning
  - array merging
  - loop interchange
- Improves temporal locality.
  - blocking

Code Positioning

- Permutes the order of the basic blocks in a program so that the most frequently executed transitions between basic blocks are fall-through transitions.
- Some processors have a penalty associated with each taken branch and this optimization can minimize this penalty.
- Also improves spatial locality, which can lower L1 IC miss rate.



(a) before positioning (b) after positioning

Array Merging

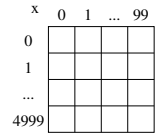
- Merges multiple arrays together as an array of structs when corresponding elements are accessed near in time to improve spatial locality.

```
/* Before */
int x[SIZE];
int y[SIZE];


for (i=0; i < 100; i++)
   sum += x[i] + y[i];
```

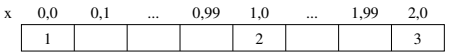
```
/* After */
struct merge {
   int x;
   int y;
} s[SIZE];

for (i=0; i < 100; i++)
   sum += s[i].x + s[i].y;
```

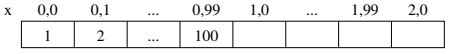

Loop Interchange

- Reorders loop iterations to improve spatial locality.




```
/* Before */
for (j = 0; j < 100; j++)
   for (i = 0; i < 5000; i++)
      x[i][j] = x[i][j]*2;

/* After */
for (i = 0; i < 5000; i++)
   for (j = 0; j < 100; j++)
      x[i][j] = x[i][j]*2;
```





Blocking

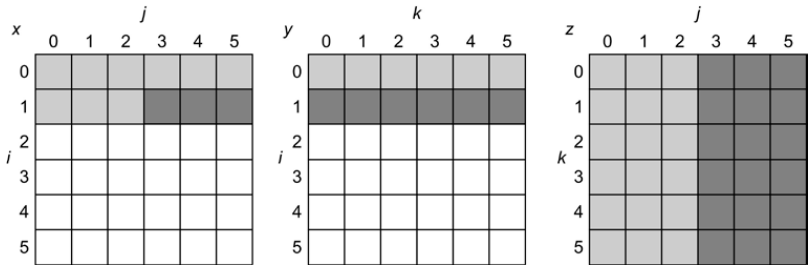
- Operate on submatrices or blocks that fit into cache.

```
for (i = 0; i < N; i++)          /* Before */
   for (j = 0; j < N; j++) {     // y[i][0] accessed again after N
      r = 0;                     // other references to y
      for (k = 0; k < N; k++)    // z[0][0] accessed again after N*N
         r += y[i][k]*z[k][j]; // other references to z
      x[i][j] = r;
   }

for (jj = 0; jj < N; jj += B)  /* After */
for (kk = 0; kk < N; kk += B)
for (i = 0; i < N; i++)
   for (j = jj; j < min(jj+B, N); j++) {  // y[i][kk] accessed again after
      r = 0;                              // B other references to y
      for (k = kk; k < min(kk+B, N); k++) // z[kk][jj] accessed again after
         r += y[i][k]*z[k][j];            // B*B other references to z
      x[i][j] += r;
   }
```


Example without Blocking

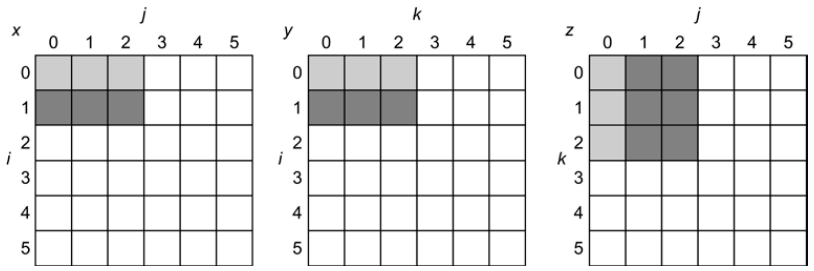
- Darker elements are more recently accessed.

```
for (i = 0; i < N; i++)          /* Before */
   for (j = 0; j < N; j++) {
      r = 0;
      for (k = 0; k < N; k++)
         r += y[i][k]*z[k][j];
      x[i][j] = r;
   }
```



Example with Blocking

```
for (jj = 0; jj < N; jj += B)  /* After */
for (kk = 0; kk < N; kk += B)
for (i = 0; i < N; i++)
   for (j = jj; j < min(jj+B, N); j++) {
      r = 0;
      for (k = kk; k < min(kk+B, N); k++)
         r += y[i][k]*z[k][j];
      x[i][j] += r;
   }
```



Blocking Summary

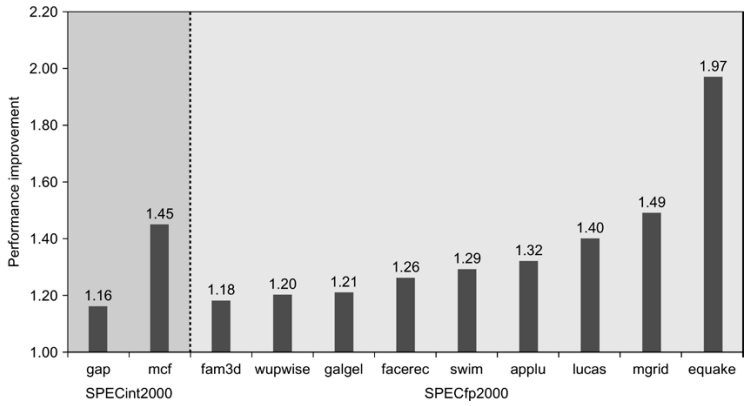
- Blocking requires more loop overhead, but better exploits temporal locality.
- Blocking also requires knowing the cache configuration for the processor, which means blocking is less portable across different implementations of the same ISA.

Hardware Prefetching

- Idea is to fetch two blocks on a miss. The referenced block is loaded into a cache. The next consecutive block is prefetched into a stream buffer.
- A valid bit is set only after the prefetched block has been loaded into the buffer, which is after the missed block was loaded into the cache.
- The stream buffers can be checked in parallel with the cache, resulting in no delay on a cache miss and a hit in the stream buffer.
- A hit in a stream buffer causes the block to be copied into the cache and another prefetch to occur.
- Prefetching exploits spatial locality.
- Can have multiple stream buffers. The stream buffer replaced on a miss is the one that is least recently accessed.

Speedup Due to Hardware Prefetching

- Graph only shows benchmarks that benefited the most from hardware prefetching.
- FP benchmarks benefit more as the data access patterns are more predictable.



Compiler-Controlled Prefetching

- Idea is for the compiler to insert instructions into the program that will initiate operations to prefetch data into the cache.
- Loops are often stripmined so that prefetch instructions are performed once for each memory block rather than once for each array element.
- Typical prefetch operation is to load data into cache, not a register, and to not fault if an exception would be generated by a conventional load from that address.

```
prefetch  0(R4)
```

- The caches have to be *nonblocking*. This means that cache hits can be serviced while the prefetched data is being loaded.
- If the hit ratio was already very high, then the execution of the prefetch instructions may degrade performance.

Applying Loop Stripmining for Prefetching

- Loop stripmining is applied that adds an outer loop to prefetch the next cache line of data.

```
/* Before */
for (p=&x; p < &x[N]; p++)
   *p = *p + s;

/* After */
for (p=&x; p < &x[N]; p += CACHESIZE) {
   prefetch(p+CACHESIZE);
   for (p2 = p; p2 < min(p+CACHESIZE, &x[N]); p2++)
      *p2 = *p2 + s;
}
```

Using High Bandwidth Memory (HBM) as an L4 Cache

- HBMs are sometimes used for very large L4 caches (128MiB to 1 GiB) instead of using HBM as a replacement for regular main memory.
- Tags and data are placed in the same HDM SDRAM row.
- First, a row access strobe (RAS) is used to get the row.
- Next, a column access strobe (CAS) is used to access the tag.
- If the tag comparison indicates a hit, then another CAS is performed to access the data, which is faster than performing another RAS.

Cache Optimization Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements

Where Does I/O Occur?

- between the I/O device and the cache
  - Solves the cache-coherency problem as the cache would always have the latest data.
  - Would interfere with the CPU.
- between the I/O device and main memory
  - Little interference with the CPU.
  - Can be performed in parallel with executing another process after a context switch.
  - Need to resolve the stale data problem.

Stale Data

- Copies of data can be in the data cache(s), main memory, and disk (virtual memory).
- *Stale data* occurs when these copies are not consistent.
- Two problems with stale data:
  - The CPU may not read the most recently input data since the cache may not be updated after an I/O input operation.
  - An I/O output operation may not have the correct output data because memory may not be up-to-date.

Solutions to Dealing with Stale Data

- Input
  - Not allow any I/O input buffers to be cachable.
  - Flush appropriate lines from the cache that are to be updated from the I/O input operation.
- Output
  - Use a write-through cache.
  - Write back all dirty lines in the cache before performing an I/O output operation.

Avoiding the Memory Wall

- Delays due to increasing memory latency have been largely avoided due to multiple optimizations.
  - multiple levels of cache
  - prefetching of data and instructions
  - compiler optimizations to improve locality
  - use of out-of-order pipelines with nonblocking caches supporting multiple outstanding misses
  - use of multithreading and thread-level parallelism

Fallacies and Pitfalls

- Fallacy: Predicting cache performance of one program from another.
- Pitfall: Simulating enough instructions to get accurate performance measures of the memory hierarchy.